

# Exact Inference Techniques for the Dynamic Analysis of Bayesian Attack Graphs

Luis Muñoz-González, Daniele Sgandurra, Martín Barrère, and Emil C. Lupu

**Abstract**—Attack graphs are a powerful tool for security risk assessment by analysing network vulnerabilities and the paths attackers can use to compromise valuable network resources. The uncertainty about the attacker's behaviour and capabilities make Bayesian networks suitable to model attack graphs to perform static and dynamic analysis. Previous approaches have focused on the formalization of traditional attack graphs into a Bayesian model rather than proposing mechanisms for their analysis. In this paper we propose to use efficient algorithms to make exact inference in Bayesian attack graphs, enabling the static and dynamic network risk assessments. To support the validity of our proposed approach we have performed an extensive experimental evaluation on synthetic Bayesian attack graphs with different topologies, showing the computational advantages in terms of time and memory use of the proposed techniques when compared to existing approaches.

**Index Terms**—Security risk assessment, attack graphs, Bayesian networks, dynamic analysis, probabilistic graphical models.

## 1 INTRODUCTION

THE estimated spending in 2015 for state-of-the art protections against cyber-attacks amounts to more than 76 billion [1]. Nevertheless, efforts to protect networks cannot cope with the sophistication of attackers, as shown by the history of data-breaches organizations have suffered over the years, including in recent times. [2]. However, it is not always possible to patch all existing vulnerabilities, since, for example, some systems cannot be interrupted or a lack of manpower prevents from doing so. Therefore, one way to optimize resources and effort required to protect a network is to firstly assess its risks, and then prioritize the most critical threats. This requires estimating the risk exposures, given the threat likelihood and the severity of the impacts [3], and then use these values to select appropriate counter-measures. The accuracy of this process produces not only a better prioritization of the threats to address, but also an improved return-on-investment. This approach, however, does not consider the dependencies between vulnerabilities, thus limiting the usefulness of the risk analysis.

These shortcomings can be addressed with Attack Graphs (AGs) [4], [5], [6], which represent prior knowledge about vulnerabilities and network connectivity, enabling system administrators to reason about threats and their risks in a formal way. In fact, AGs permit a priori analysis of the possible avenues an attacker can exploit to compromise the system. Hence, they can be used to focus on the most-effective threats and produce a better selection of counter-measures [7], which is also known as *static analysis*.

On the other hand, proactive security hardening is not always the best strategy. As discussed in [8], reactive security can be competitive with proactive security when the reactive defender does not overreact to the last attack but learns from past experience. In this sense, AGs can also be used to *dynamically* profile the attacker's paths, to understand which

nodes are more likely to be attacked in the next steps. They can also be used to evaluate the security risk for valuable resources in the network and reason about the nodes that may be already compromised when we observe evidence of an ongoing attack. Since organizations are often under attack, this dynamic analysis gives system administrators important insights in understanding where they should spend their efforts in real-time, and what are the most vulnerable targets.

Both *static* and *dynamic analysis* of AGs have inherent probabilistic characteristics given the uncertainty about the attackers' ability to successfully exploit vulnerabilities. In this sense, Bayesian Networks (BNs) provide an appropriate framework to model AGs since they depict causal relationships between random variables in a compact way. This approach has already been proposed in the literature on AGs. [9] present a Bayesian AG (BAG) to model potential attack paths in a network, using Variable Elimination (VE) as an algorithm for inference on the Bayesian model. [10], [11] present mechanisms to calculate the conditional probability tables, which represent the combined effect of vulnerabilities to compromise a node in BAG models. More recently, [12] present a BN framework to perform risk assessment and propose risk mitigation strategies in the context of AGs.

However, none of the above propose appropriate and efficient algorithms for inference on their models, i.e., to calculate the probabilities that an attacker can compromise the network nodes. Note that computing unconditional probabilities in BNs is an NP-Hard problem. Therefore, the use of efficient inference techniques is important to reduce the time and computational resources required and improve the applicability of BNs for the static and dynamic analysis of AGs. More concretely, in [10], [11] no mechanism is proposed to calculate the unconditional probabilities of compromising each node. Forward-backward propagation is proposed to perform inference in [12]. However, as shown in [13], [14], this technique is only valid when the corresponding graph is a chain, which is not true for AGs in

• All the authors are with the Department of Computing at Imperial College London, 180 Queen's Gate, SW7 2AZ, London, UK.  
E-mail: {l.munoz, d.sgandurra, m.barrere, e.c.lupu}@imperial.ac.uk

general. Finally, although the VE algorithm proposed in [9] is valid for inference in BAGs, its computational complexity limits its applicability to small graphs, especially in the case of the dynamic analysis where the time to respond to an attack is of essence. It is also worth noting that none of the previous papers reports an experimental evaluation of the time and memory required by the techniques proposed to assess their suitability for static and dynamic analysis of AGs.

The main contributions of this paper are the following:

- We propose a revised BAG model valid for the static and dynamic analysis of AGs. This model aims to overcome some limitations of previous models, such as the undesirable effects of adding a prior on the attacker capabilities, and to serve as a core framework for further model extensions, such as zero-day vulnerabilities, attacker's capabilities, real behaviour of Intrusion Detection Systems (IDSs), or dependences between vulnerabilities, among others.
- Although exact inference in probabilistic graphical models is NP-Hard, we propose to use message passing algorithms such as Belief Propagation (BP) (for inference Attack Trees) and Junction Tree (JT) (for general AGs), to efficiently calculate the unconditional probabilities that the nodes have been compromised considering, if applicable, evidence of an ongoing attack.
- To assess the applicability of the proposed algorithms, and to show limitations of existing approaches, we provide a comprehensive experimental evaluation using synthetic AGs. Our results show that the JT algorithm can be applied to AGs of hundreds of nodes, corresponding to networks of thousands of nodes. As far as we know, this is the first experimental evaluation in the literature of AGs that analyses the time and memory requirements for static and dynamic inference in BAGs.
- Our results also show the importance of clustering when modelling with AGs. We show that the JT algorithm in clustered networks scales linearly in the number of nodes for dynamic inference, which makes it suitable for use in practical settings. Fortunately, clustering occurs in most corporate networks where hosts are grouped e.g., into different sub-networks, for management and networking.

The rest of the paper is organised as follows. In Section 2 we review the concept of AGs. In Section 3 we present a BAG model that considers and improves upon existing models in the literature. In Section 4 we present Variable Elimination, Belief Propagation, and Junction Tree as procedures to perform exact inference on BAGs and Bayesian Attack Trees. Experimental results on static and dynamic inference of AGs are shown in Section 5. In Section 6 we sketch some possible extensions of the BAG model. Finally, in Section 7 we present the main conclusions and further research directions.

## 2 ATTACK GRAPHS

AGs are graphical models that depict the knowledge about vulnerabilities, and their interaction, in a network, by show-

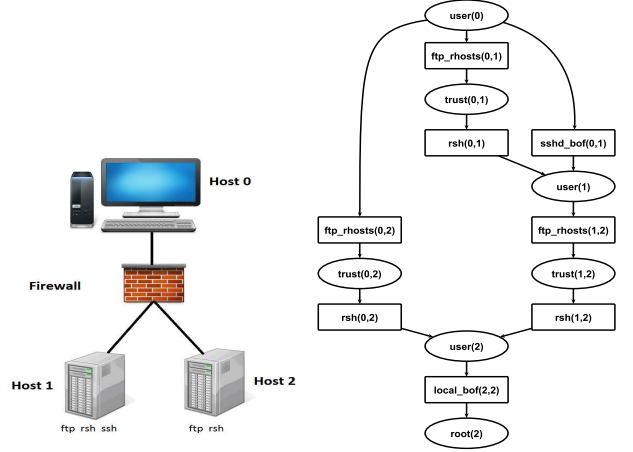


Fig. 1. Simple example of a network configuration and the corresponding logical AG taken from [21], [22].

ing different paths that an attacker can take to reach a given goal. This is possible by exploiting vulnerabilities in sequence, each successful exploit giving the attacker more security privileges towards the goal. Two main types of AG representations are used in the literature, namely *state-based representations* and *logical AGs*.

In *state-based* representations [4], [5], [15], [16], [17] each node in the AG depicts the state of the whole network after a simple atomic attack, and contains a table with global variables defining the state of the complete network. This limits the practical application of this kind of AGs to very small networks [18], [19], [20] due to the combinatorial explosion in the number of states and variables when increasing the number of nodes. Moreover, state-based AGs can contain duplicate attack paths that differ only in the order of the attack steps. This also contributes to increase the complexity of the graph.

In contrast, *logical AGs* can be defined as bipartite graphs which depict the dependencies between exploits and security conditions [18]. These representations rely on the concept of monotonicity: The attacker never relinquishes privileges once obtained. Although this is not completely realistic under some specific attacks, it is in most cases a valid assumption. Monotonicity allows to remove duplicated paths and results in a Directed Acyclic Graph (DAG), which grows polynomially with the number of vulnerabilities and the number of connected pairs of hosts [6]. Formally, we can define an AG (in the logical representation) as a directed bipartite graph  $G = (E \cup C, R_r \cup R_i)$ , where the vertices  $E$  and  $C$  are the sets of exploits and security conditions, respectively, and the edges  $R_r \subseteq C \times E$  and  $R_i \subseteq E \times C$  are *require* and *imply* relations.

Figure 1 shows a simple scenario, taken from [21], [22], where the Host 1 offers File Transfer Protocol (FTP), Secure Shell (SSH), and Remote Shell (RSH), whilst Host 2 offers FTP and RSH. The firewall allows FTP, SSH, and RSH traffic from external users (Host 0) to both servers. The goal of the attacker is to gain root privileges on Host 2. In the AG, the conditions are represented as circles where the host involved is inside the parentheses, while vulnerabilities are depicted in rectangles, indicating the source and destination host

inside the parentheses, i.e. (*source, destination*). In Figure 1, it can be appreciated that the attacker can follow three possible paths to achieve his goal. The probabilities that an attacker can successfully exploit the vulnerabilities in the network given in [22], using the Base Score metric of the CVSS score, are: 0.8 for *ftp\_rhost*, 0.1 for *ssh\_bof*, 0.9 for *rsh*, and 0.1 for *local\_bof*. Note that these probabilities do not take into account the attacker's capabilities and estimate only their average probability of successful exploitation. Modelling the attacker's capabilities is beyond the scope of this paper.

### 3 BAYESIAN ATTACK GRAPHS

As logical representations of the AGs result in DAGs, Bayesian Networks (BNs) are suitable to model the AGs and perform static and dynamic analysis, thus allowing to calculate the probability of an attacker to reach each state (security condition) in the graph.

The use of BNs for AGs was first introduced in [9] for the dynamic analysis of AGs. The authors proposed using the VE algorithm [23] to calculate the probability that an attacker can reach a security state given prior knowledge of the current state reached by the attacker. They also propose to use the Most Probable Explanation (MPE) algorithm (relying on VE) to compute the nodes that the attacker has possibly already compromised. However, VE can be computationally expensive compared to other inference algorithms such as Junction Tree (JT). Furthermore, the authors do not propose an elimination ordering algorithm before applying VE, which has significant impact on the algorithm's performance, as we show in Section 5. Finally, we consider that the use of MPE is not appropriate in the context of AGs and can lead to misleading conclusions about the network state of the, as further discussed in Section 4.

[11] and [10] show how to calculate the conditional probability tables in a Bayesian Attack Graph (BAG) as the combined effect of vulnerabilities in a network. In [21], a Dynamic BN is proposed to also model temporal factors that affect the impact of the vulnerabilities, however they do not provide any mechanism to make inference in their models. [12] proposes a risk management framework using BNs to assess the chances of network compromise at run-time and mitigation strategies with AGs using the probabilities calculated by the Bayesian model. However, the authors propose to use forward-backward propagation for inference in the model, which is not appropriate for general AGs, as this algorithm can only be applied to chains, not to general graphs [13], [14].

A BN is a directed graphical model where the nodes represent random variables and the directed edges represent the dependencies between random variables, forming a DAG. Let  $\mathbf{X} = \{X_1, \dots, X_n\}$  be a set of random variables (continuous or discrete). The joint probability distribution can be written as:

$$p(\mathbf{X}) = \prod_{i=1}^n p(X_i | \mathbf{pa}_i) \quad (1)$$

so that, under the BN representation, for each node  $X_i$  there is a directed edge from each node in the set of parents nodes  $\mathbf{pa}_i$  of  $X_i$  pointing to  $X_i$ . For example, the joint probability

distribution of the BAG depicted in Figure 2 can be written as:

$$p(A, B, C, D, E, F, G) = p(A) p(B|A) p(D|A) p(C|A, B) \times p(E|C) p(F|D, E) p(G|F) \quad (2)$$

#### 3.1 Model assumptions

In this paper, we will make the following assumptions to build the BAG from the standard logical representation: 1) There are no dependencies between vulnerabilities: Successful exploitation of one vulnerability by an attacker does not change the probability of successfully exploiting other vulnerabilities. 2) The probability of successful exploitation of a vulnerability is assumed to be constant in time and does not depend on the attacker's skills. Although in [21] a dynamic network is proposed to model the dynamic aspects of vulnerabilities, the potential changes in these probabilities are slow enough (they can change in the scope of days or weeks) to be considered constant. Then, it is better to recompute the model when there are changes, than to increase the complexity of the model to deal with the dynamic aspects of the probability of exploitation of vulnerabilities. 3) The attacker's capabilities are not considered or, at least, all the potential attackers are supposed to have the same skills and attack preferences. 4) We do not consider zero-day vulnerabilities, social engineering attacks, and insider attacks. 5) The Intrusion Detection System (IDS) does not trigger false alarms. Although we assume that some events may be stealthy, i.e. not detected by the IDS.

Although the assumptions can seem restrictive, they are common in the literature of AGs. The aim of this paper is to propose efficient probabilistic inference mechanisms for the dynamic analysis of AGs. Nevertheless, the inference algorithms proposed in this paper can be applied to other extended and more flexible BAG models, as discussed in Section 6.

Under these assumptions, the nodes in the BAG represent the different security states that an attacker can reach. We model the behaviour of these states as Bernoulli random variables, so that the probability of a node  $X_i$  to be compromised is<sup>1</sup>  $\Pr(X_i = T) = p$ , and, consequently, the probability of a node not to be compromised is  $\Pr(X_i = F) = 1 - p$ , with  $p \in [0, 1]$ .

In Figure 2 we show the BAG generated from the standard AG depicted in Figure 1 along with the probabilities for each node to be compromised by an attacker. In the BAG, the initial node,  $A \equiv \text{user}(0)$ , represents that the attacker has user privileges on his own machine with probability 1.

#### 3.2 Conditional Probability Distributions

Under the BN representation, the information available at each node is the conditional probability distribution of a node to be compromised given its parent nodes, i.e.,  $p(X_i | \mathbf{pa}_i)$ . From a security view point, these conditional probabilities represent the probabilities of an attacker to reach a security state  $X_i$  given the observations of the set

1. In the following, to simplify the mathematical notation, we will refer to the unconditional probability of a node to be compromised as  $\Pr(X_i)$ .

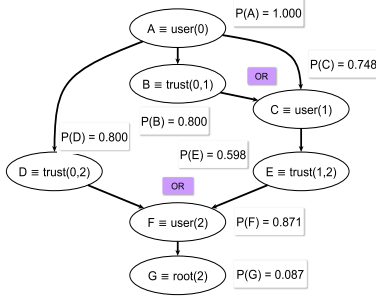


Fig. 2. BAG representation for the AG in Figure 1 with the unconditional probabilities calculated for each node when there are no attacks.

of preconditions  $\mathbf{pa}_i$  that allow the attacker to compromise  $X_i$  by exploiting the vulnerabilities  $e_i$ . These vulnerabilities are those that link  $\mathbf{pa}_i$  with  $X_i$  in the original bipartite AG. We consider that the probabilities of successfully exploiting vulnerabilities are parameters of the model (instead of random variables) which allow to calculate the conditional probability tables that define  $p(X_i|\mathbf{pa}_i)$ .

The scores provided by the Common Vulnerability Scoring System (CVSS) [24] can be used to estimate  $p_{v_j}$ , the probability of an attacker successfully exploiting a vulnerability  $v_j$ . Although CVSS scores are intended to estimate the impact of a vulnerability rather than its probability of it being successfully exploited, in the absence of other better indicators, CVSS scores or some of their submetrics are often used in the literature e.g., [12], [21], [25] as an acceptable estimate. Whilst [12], [21], [25] use the entire CVSS score, in our opinion, the exploitability submetric is more appropriate since it tries to measure the difficulty of exploiting a vulnerability. This is also proposed in [12].

For the AG in Figure 1 we have used the probabilities of exploitation given by [22], which only consider the Base Score metric of the CVSS score.

To calculate the conditional probability distributions  $p(X_i|\mathbf{pa}_i)$  given the probabilities of successfully exploiting the vulnerabilities that allow an attacker reach a security condition from  $\mathbf{pa}_i$ , we consider two possible cases [12]: A logical AND where all the preconditions should be met in order to compromise node  $X_i$ . This can be expressed as:

$$p(X_i|\mathbf{pa}_i) = \begin{cases} 0, & \exists X_j \in \mathbf{pa}_i | X_j = F \\ \prod_{j: X_j \in \mathbf{pa}_i} p_{v_j}, & \text{otherwise} \end{cases} \quad (3)$$

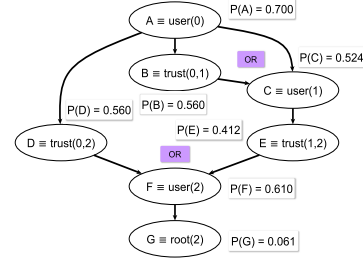
A logical OR where only one of the preconditions in  $\mathbf{pa}_i$  needs to be satisfied to execute an exploit and compromise node  $X_i$ . This can be calculated using the noisy-OR formulation [26], so that:

$$p(X_i|\mathbf{pa}_i) = \begin{cases} 0, & \forall X_j \in \mathbf{pa}_i | X_j = F \\ 1 - \prod_{j: X_j \in \mathbf{pa}_i} (1 - p_{v_j}), & \text{otherwise} \end{cases} \quad (4)$$

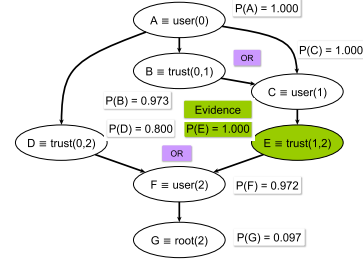
In the Supplementary Material we show through an example how to compute the conditional probability tables for the AND and OR cases for a random variable with three parent nodes.

### 3.3 Effect of the prior probability on the initial state

In this subsection we discuss the effect of the initial node of the BAG, which represents the initial state of the attacker



(a)



(b)

Fig. 3. (a) Unconditional probabilities for the BAG in Figure 2 when the prior belief on the initial state of the attacker (node A) is set to 0.7. (b) Unconditional probabilities for the previous BAG when evidence that the attacker has compromised node E is observed. The result is the same than in the case of observing the same evidence for the BAG in Figure 2.

(for example, node A in Figure 2). In our opinion, this node does not represent a random variable, since it only represents that the attacker has full rights on his own machine. This is equivalent to consider that  $\Pr(X_0) = 1$ , where  $X_0$  is a Bernoulli random variable that models the initial state of the attacker, although  $X_0$  is not really random, since it can take only the value T. This contrasts with other approaches that consider  $X_0$  as a random variable.

Concretely, in [12], they propose to use this node to reflect some subjective prior knowledge of the attacker capabilities by letting the administrator set the value of  $\Pr(X_0)$ . This has two main shortcomings: Firstly, modelling the attacker's capabilities only describes a subjective average behaviour of different kinds of attackers and secondly, the effect of this prior can lead to misleading conclusions in the dynamic analysis of the AG, especially when reasoning using new evidence about nodes that an attacker may have already compromised.

To illustrate this let us use the example shown in Figure 3.(a), where we have the same BAG as in Figure 2, but with the prior belief on the initial state to the attacker set to  $\Pr(A) = 0.7$ , instead of 1. As expected, the unconditional probabilities of the other nodes decrease with respect to the probabilities calculated in Figure 2. However, once we observe some evidence of an attack and recompute the unconditional probabilities, as shown in Figure 3.(b), we observe that an attacker has compromised node E. The result is the same regardless of the value of  $\Pr(A)$ . But analysing the differences between the unconditional probabilities before and after observing the attack, for the case where  $\Pr(A) = 1$ , all nodes have increased their probabilities except D. This indicates that the attacker has compromised node C and

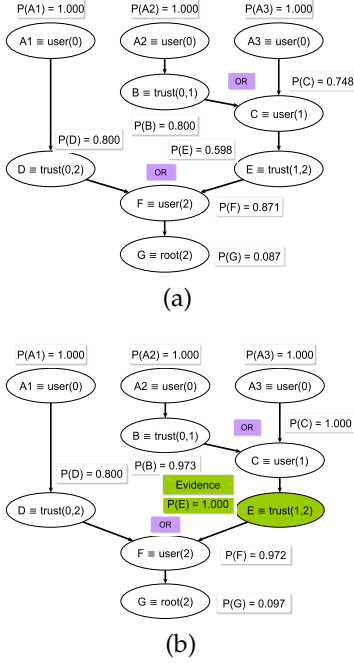


Fig. 4. (a) Unconditional probabilities for the BAG in Figure 2 splitting the node of the initial state of the attacker in 3 nodes, one for each initial attack path. (b) Unconditional probabilities for the previous BAG when evidence that the attacker has compromised node  $E$  is observed. The result is the same than in Figure 3.(b).

possibly node  $B$ . In contrast, if  $\Pr(A) = 0.7$  we observe that, given the evidence of attack at node  $E$ , the probabilities of all the nodes have increased. This can be misleading, since there is no reason to think that the attacker followed the path from  $A$  to  $D$  or that the attacker capabilities to compromise  $D$  have increased. This effect can hinder reasoning about the possible attack paths that the attacker could follow or the nodes that may have been already compromised.

Finally, to calculate the unconditional probabilities of the BAG more efficiently, we propose to use one initial node (the initial state of the attacker) for each possible initial attack path. This does not affect the value of the unconditional probabilities of the rest of the nodes (with and without evidence of any possible attack), but it allows to break some loops in the graph which reduce the complexity of the inference algorithms described in the following sections. For example, following these guidelines, the BAG in Figure 2 becomes a tree, as shown in Figure 4, which makes it suitable to use BP to efficiently calculate the unconditional probabilities of all the nodes. It is also interesting to note that splitting the initial state of the attacker in different nodes, even if the prior belief on the attacker capabilities is not set to 1, we can track the possible paths that the attacker followed when we observe evidence, since, under this assumption, all the possible initial attack paths are now considered independently.

#### 4 EXACT INFERENCE IN BAYESIAN ATTACK GRAPHS

For the analysis of AGs, we are interested in calculating the unconditional probability distributions  $p(X_i)$ , rather than  $p(X_i|\mathbf{pa}_i)$ , to calculate the probability that an attacker

can reach a given security condition, and thus, the risk. Using Bayes rule it is possible to calculate  $p(X_i)$  from the conditional probability distributions:

$$p(X_i) = \sum_{\mathbf{X}-X_i} p(\mathbf{X}) = \sum_{\mathbf{X}-X_i} \prod_{j=1}^n p(X_j|\mathbf{pa}_j) \quad (5)$$

where  $\mathbf{X} - X_i$  indicates that we sum over all the set of random variables  $\mathbf{X}$  except  $X_i$ .

However, the exact calculation of (5) is an NP-Hard problem [26], [27]. In our case, as each node corresponds to a Bernoulli random variable, the memory required to store the joint probability distribution  $p(\mathbf{X})$  grows as  $2^n$ . For example, the joint probability distribution for a BAG with 40 nodes, using 8 bytes to store each entry in the table, requires  $2^{40+3}/1,024^3 = 8,192$  Gigabytes of memory. Thus, applying brute force for inference in probabilistic graphical models is not a reasonable approach in terms of computational time and memory, even for small graphs and the use of efficient algorithms is a strong requisite.

In the literature on BAGs this issue has not been covered adequately. In [10], [21], [22] static and dynamic BAG models are proposed, describing how to calculate the conditional probability tables, but the authors do not propose any method to compute the unconditional probabilities. [12] proposes to use forward-backward propagation. However, this procedure is only valid for chains [13], [14] and cannot be applied to general AGs. Finally, the VE algorithm [23] is proposed for inference in [9]. Although this is an efficient technique to calculate the unconditional probabilities, the authors in [9] do not propose any heuristic to find a reasonable elimination ordering for VE. This issue impacts significantly the performance of the algorithm and, moreover, finding the optimal elimination order turns out to also be an NP-Hard problem [28]. [9] also lacks an experimental evaluation to assess the validity and scalability of the algorithm in the context of AGs.

VE is also used in [9] for Maximum A Posteriori estimation to provide the MPE. However, these MAP estimations can lead to misleading conclusions in the context of AGs. For the example in Figure 2, the result of the MPE queries when there is no evidence of attacks is that all the nodes except  $G$  are in the True state (and then  $G$  is in the False state). The attacker would have therefore already compromised all the nodes except  $G$ , which makes no sense. In this case we can easily show that MPE is useless to assess the security risk in AGs.

In the next section, we first review VE algorithm, explaining how it can be applied for inference on BAGs and then, we describe the BP and JT algorithms, which efficiently calculate the unconditional probabilities on BNs using a message passing approach. This reduces significantly the average computational complexity with respect to VE.

##### 4.1 Variable Elimination

VE or Bucket Elimination was first introduced by [29] and revisited in [23] as a heuristic to efficiently compute the unconditional probabilities in BNs and Markov Random Fields (MRFs). In essence, to address the exponential blow-up when computing the marginal probabilities it looks for factors in the joint distribution that depend only on a small



number of variables, computes them once and then caches the results to avoid generating them exponentially many times [26].

For example, consider the probability  $p(G)$  that an attacker can obtain root privileges on *Host 2*, in the AG shown in Figure 2. We can then write  $p(G)$  as:

$$p(G) = \sum_A \sum_B \sum_C \sum_D \sum_E \sum_F p(A) p(B|A) p(C|A, B) \times \\ \times p(D|A) p(E|C) p(F|D, E) p(G|F) \quad (6)$$

As discussed before, computing the joint distribution scales in time and memory as  $\mathcal{O}(2^n)$ , with  $n = 7$  in this case. In contrast to this brute force approach, VE proceeds by grouping factors that involve the same variables and marginalizing (summing over) those variables. Then, we can re-write  $p(G)$  as:

$$p(G) = \sum_F p(G|F) \sum_E \sum_D p(F|D, E) \sum_C p(E|C) \times \\ \times \sum_B \sum_A p(A) p(B|A) p(C|A, B) p(D|A) \quad (7)$$

Evaluating this expression from right to left we can recursively eliminate all the variables in the BN except  $G$ . In this case, we follow the elimination ordering  $\Omega = \{A, B, C, D, E, F\}$ . The steps of elimination using the VE algorithm are shown in Table 1: At each step we create a new factor  $\phi_i$  by multiplying all the factors that involve the variable we want to eliminate, and then, we marginalize the corresponding variable from the factor  $\phi_i$ .

TABLE 1  
Steps of VE algorithm for the BAG in Figure 2 to calculate  $p(G)$  using the elimination order  $\Omega = \{A, B, C, D, E, F\}$ .

Var.	Factors
A:	$\phi_1(A, B, C, D) = p(A)p(B A)p(C A, B)p(D A)$ $\tau_1(B, C, D) = \sum_A \phi_1(A, B, C, D)$
B:	$\phi_2(B, C, D) = \tau_1(B, C, D)$ $\tau_2(C, D) = \sum_B \tau_1(B, C, D)$
C:	$\phi_3(C, D, E) = \tau_2(C, D)p(E C)$ $\tau_3(D, E) = \sum_C \phi_3(C, D, E)$
D:	$\phi_4(D, E, F) = \tau_3(D, E)p(F D, E)$ $\tau_4(E, F) = \sum_D \phi_4(D, E, F)$
E:	$\phi_5(E, F) = \tau_4(E, F)$ $\tau_5(F) = \sum_E \phi_5(E, F)$
F:	$\phi_6(F, G) = \tau_5(F)p(G F)$ $\mathbf{p}(\mathbf{G}) = \tau_6(G) = \sum_F \phi_6(F, G)$

The same principle applies when we observe evidence of compromise on some of the nodes and we want to compute the posterior probabilities of the nodes given the evidence. In this case, as described in [13], [26], we first compute the joint probability distribution of the query variable and the evidence, and then, divide by the marginal probability of the observed evidence. For example, if we observe that an attacker has compromised node  $C$  in Figure 2 (the attacker has user privileges on Host 1), the posterior probability of  $G$  given the evidence is then calculated as:

$$p(G|C = 1) = \frac{p(G, C = 1)}{p(C = 1)} \quad (8)$$

The computational cost of the algorithm is exponential in the scope of the biggest factor created during elimination, i.e., the factor with the maximum number of variables. In the example shown in Table 1, the biggest factor is  $\phi_1$ , whose scope is  $A, B, C, D$ . This requires to compute a table with  $2^4 = 16$  entries, whereas the computation of the expression in (6) requires a table with  $2^7 = 128$  entries. So, even in this simple example we can notice significant savings.

The variables' elimination order has a significant impact on the size of the intermediate factors created, and thus, in the computational complexity of the algorithm [26], [30]. For instance, if in the previous example we apply the alternative elimination ordering  $\Omega' = \{B, A, C, D, E, F\}$ , the factor created to first eliminate  $B$  is  $\phi'_1(A, B, C) = p(A) p(B|A) p(C|A, B)$ , so that  $\tau'_1(A, C) = \sum_B \phi'_1(A, B, C)$ . Then,  $\phi'_2(A, C, D) = p(A) p(C|A, B) \tau'_1(A, C)$  and  $\tau'_2(C, D) = \sum_A \phi'_2(A, C, D)$ . The rest of the elimination steps are the same as those in Table 1. With this elimination ordering the maximum scope of the biggest factor is reduced from 4 to 3.

Although finding the elimination order that minimizes the scope of the biggest factor is also NP-Hard [26], several greedy heuristics [26], [30], [31], [32] provide reasonably good elimination orders at reduced computational cost. These rely on the concept of an *induced graph*: The graph obtained when eliminating a variable from the original one. They aim to provide orderings that induce small graphs, which corresponds to eliminating variables so that the scope of the intermediate factors  $\phi_i$  and  $\tau_i$  remains as small as possible. Some criteria that have been commonly used for this task are [26], [30]: 1) *Min-neighbours*: At each step of VE elimination we eliminate the node with the fewest number of neighbours in the current graph; 2) *Min-fill*: At each step we eliminate the node whose removal implies adding the smallest number of edges in the induced graph; 3) *Min-weight*: At each step we remove the node with the minimum product of weights of its neighbours, where the weights are the number of elements in the scope of the conditional probability associated to the node (for example, in the BAG in Figure 2, the weight for node  $E$  would be  $3 \times 3 = 9$ , since its neighbours  $C$  and  $F$  have 3 variables in their scope); 4) *weighted-min-fill*: At each step we eliminate the node with the smallest sum of weights of the edges that need to be added to the graph due to its elimination, where the weight of an edge is the product of weights of the corresponding nodes associated to that edge.

The use of the VE algorithm for BAGs was previously proposed in [9]. However, the authors use the algorithm from [23], where no elimination ordering is proposed. In the experiments described in Section 5, we will show the impact of the elimination ordering in terms of the time and memory required to compute the unconditional probabilities.

Finally, as discussed in [13], the main disadvantage of VE, in addition to the exponential scalability with the scope of the biggest factor, is its inefficiency when we compute multiple queries, e.g. when we want to calculate the unconditional probabilities of all the nodes in the graph. In VE we need to compute the elimination ordering and the factors each time we make a query, whereas other algorithms like BP or JT cache information (messages) that can be re-used to efficiently compute the marginal probabilities in a BN or

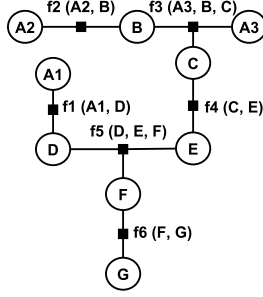


Fig. 5. Factor graph representation for the BAG in Figure 4.

a MRF as we explain next.

## 4.2 Belief Propagation

Like VE, BP is an algorithm to efficiently compute the unconditional probabilities in BNs and MRFs when the structure of the graph is a tree or a polytree. Although this is not the general structure of AGs, this technique can be applied to Attack Trees (ATs) [33]. BP is also referred as the sum-product algorithm and it is based on probabilistic message passing. This also is the basis of the JT algorithm which can be applied to any kind of BN and that we describe in Section 4.3. The first version of BP for trees was first proposed in [34] and, then, extended in [35] for the case of polytrees, although the complete formulation of BP is only introduced in [36].

To describe the algorithm we follow a similar approach as in [37], using factor graph [38], [39] representations. As explained before BNs (and MRFs) allow to express the joint probability distribution of several random variables as the product of factors over subsets of those variables. Factor graphs make this decomposition more explicit by introducing additional nodes for the factors themselves in addition to the nodes representing the random variables, resulting in a bipartite graph.

For example, the joint probability of the BAG shown in Figure 4 can be expressed as:

$$p(A_1, A_2, A_3, B, C, D, E, F, G) = \prod_{i=1}^6 f_i(\mathbf{X}_i) \quad (9)$$

where the factors  $f_i(\mathbf{X}_i)$  are:

$$\begin{aligned} f_1(A_1, D) &= p(A_1) p(D|A_1) \\ f_2(A_2, B) &= p(A_2) p(B|A_2) \\ f_3(A_3, B, C) &= p(A_3) p(C|A_3, B) \\ f_4(C, E) &= p(E|C) \\ f_5(D, E, F) &= p(F|D, E) \\ f_6(F, G) &= p(G|F) \end{aligned} \quad (10)$$

The corresponding factor graph representation is shown in Figure 5. It is important to note that for a given BN or MRF there can exist more than one factor graph.

BP works by passing real valued functions called messages among neighbouring nodes in the tree or polytree network. Since the factor graph induces a bipartite graph, we can distinguish two different types of messages: From variable to factor and from factor to variable. Messages from

a variable node  $X_i$  to a factor  $f_j$  (in the neighbourhood of  $X_i$ ) are given by:

$$\mu_{X_i, f_j}(X_i) = \prod_{f_k \in \{\mathbf{F}_i - f_j\}} \mu_{f_k, X_i}(X_i) \quad (11)$$

where  $\mu_{f_k, X_i}(X_i)$  are the messages from the set of factor nodes in the neighbourhood of  $X_i$  except  $f_j$ . On the other hand, messages from a factor node  $f_i$  to a variable node  $X_j$  in the neighbourhood of  $f_i$  are calculated as:

$$\mu_{f_i, X_j}(X_j) = \sum_{X_k \in \mathbf{X}_s} f_i(X_j, \mathbf{X}_s) \prod_{X_k \in \mathbf{X}_s} \mu_{X_k, f_j}(X_k) \quad (12)$$

where  $\mathbf{X}_s$  is the set of variable nodes in the neighbourhood of  $f_i$  except  $X_j$ .

When a variable  $X_i$  is a leaf node, the corresponding messages to the factors in its neighbourhood are equal to one, i.e.  $\mu_{X_i, f_j}(X_i) = 1$ . On the contrary, if a factor  $f_i$  is a leaf node in the graph, messages to a variable node in its neighbourhood are  $\mu_{f_i, X_j}(X_j) = \sum_{X_k \in \mathbf{X}_s} f_i(X_j, \mathbf{X}_s)$ .

BP computes all the messages from all node variables to their corresponding factors and vice versa starting from the leafs nodes and propagating the messages across the tree or polytree graph according to the following rule: A node  $N$  (variable or factor node) cannot send a message to other node  $M$  until  $N$  receives all messages from its neighbour nodes except  $M$ .

An example of the message passing process of BP for the factor graph in Figure 4 is shown in the Supplementary Material.

Once all messages are computed, the unconditional probability for a node  $X_i$ , provided that the graph is a BN, can be calculated as:

$$p(X_i) = \prod_{f_j \in \mathbf{F}_i} \mu_{f_j, X_i}(X_i) \quad (13)$$

where  $\mathbf{F}_i$  are the factor nodes in the neighbourhood of  $X_i$ . In the case of MRFs, the same principle applies, but the product of the messages results in an unnormalized version of  $p(X_i)$ , though the calculation of the normalizing constant is straightforward.

In contrast to VE, where we need to run the whole algorithm for each marginal probability that we compute, BP can efficiently calculate all the marginal probabilities by computing all the messages once and storing them.

So when we observe evidence of compromise in some of the graph's nodes, only the factors that depend on the values that have changed need to be re-computed. For example, if the attacker has compromised node  $D$ , i.e. he has acquired a trust relation with Host 2 from his machine, when computing messages involving factor  $f_1$  we should take into account only the values of the function for which  $D = T$ , so that

$$f_1(A_1, D = T) = p(A_1) p(D = T|A_1) \quad (14)$$

which requires only to consider the elements of the conditional probability table for  $p(D|A_1)$  where  $D = T$ . On the other hand, when computing the messages from factors to variable nodes involving observed variables, i.e. variables for which we observe evidence, we do not need to sum over these observed variables to obtain the corresponding message.

The details about the computational complexity of BP will be discussed in Section 4.3, since BP can be considered a special case of the JT algorithm.

### 4.3 Junction Tree

In this section we describe the JT or clique tree algorithm, a method that takes the advantage of the message passing scheme of BP to compute all the marginals of a BN or a MRF, but applied to general graphs like in the case of VE rather than just trees and polytrees. The aim of JT is to create a tree structure where the nodes represent clusters of the random variables in the graph, and then, apply message passing as in BP, to compute the unconditional probabilities. This method is equally applicable to both BNs and MRFs.

In [36], besides presenting BP algorithm for polytrees, a simple approach to cluster nodes in general graphs is also described. However, the technique proposed produces very inefficient trees [26]. In contrast, following the lines of BP, the two variants of JT algorithm, namely the Shenoy-Shafer algorithm [40], [41] and the Hugin algorithm [42], [43] produce more efficient trees by clustering nodes. Both techniques rely on the same principles although they differ in the way the messages are computed. In the following, we will use the Shenoy-Shafer method which uses the same message passing scheme that we described for BP. A detailed comparison of Hugin and Shenoy-Shafer can be found in [44].

The first step of the JT algorithm is to create a cluster graph with a tree structure from the initial BN (or MRF). This can be viewed as an extension of factor graphs that clusters several random variables between two factors where each random variable can appear in more than one cluster node. As a requisite for the JT algorithm, the cluster or clique tree must also satisfy the running intersection property: If a random variable  $X_i$  appears in two cluster nodes,  $X_i \in C_j$  and  $X_i \in C_k$ , then,  $X_i$  also appears in each cluster node in the unique path existing between  $C_j$  and  $C_k$  in the clique tree.

As shown in [26], an execution of VE induces a cluster graph with a tree structure that satisfies the running intersection property. There are also other procedures that rely on creating a chordal graph by moralizing and triangulating the original BN or MRF, so that each clique in the chordal graph is a cluster node in the clique tree. However, for the sake of clarity and since both solutions have a similar computational burden, we prefer to describe the use of VE as a method to obtain the clique tree.

To describe the procedure of generating the clique tree we use the BAG shown in Figure 2 and the corresponding execution of VE following the elimination order  $\Omega = \{A, B, C, D, E, F\}$  shown in Table 1. First of all, we create a initial factor  $f'_i$  for each  $\phi_i$  used in the computation of VE. Then, we draw an edge between  $f'_i$  and  $f'_j$  if the factor generated from  $\tau_i$  is used in the computation of  $\tau_j$ . In our example, we have an edge between the factors  $f'_1(A, B, C, D)$  and  $f'_2(B, C, D)$ , corresponding to the terms  $\phi_1(A, B, C, D)$  and  $\phi_2(B, C, D)$  respectively, because  $\phi_2(B, C, D)$  depends on  $\tau_1(B, C, D)$ . Next, we add cluster nodes between each pair of factors considering all the random variables that intersect the two adjacent factors. For example, between

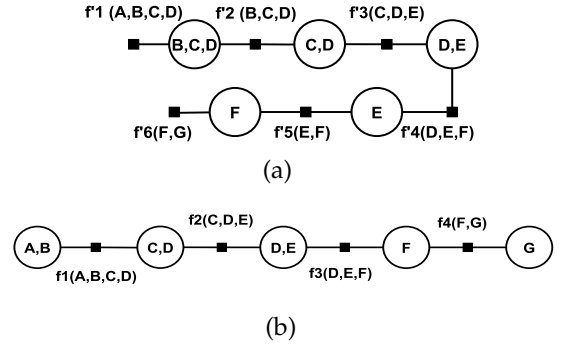


Fig. 6. (a) Initial factors for the JT of the BAG in Figure 2 following the elimination order in Table 1. (b) Final factors for the JT after clustering factors and adding the leaf variable nodes.

$f'_1(A, B, C, D)$  and  $f'_2(B, C, D)$  we include a cluster node with the variables  $B, C, D$ . After these two steps we get the JT shown in Figure 6.(a). However, we can reduce the tree by removing redundant factors i.e., those whose scope is a subset of the scope of adjacent factors, removing also the corresponding cluster nodes between the implied factors. This is the case of factors  $f'_2$  and  $f'_5$  in our example, whose scopes are a subset of the scopes of  $f'_1$  and  $f'_4$  respectively. Finally, we add to each leaf factor node a cluster node, which becomes a leaf node, with the random variables that are in the scope of that factor but are not in the other cluster nodes associated to it. For example, for  $f'_1$  we should add a cluster node with the variables  $A$  and  $B$ , since the other cluster node associated to  $f'_1$  contains  $C$  and  $D$  and the scope of  $f'_1$  is  $\{A, B, C, D\}$ . The reduced final clique tree is shown in Figure 6.(b).

Once we have defined the factors and obtained the reduced JT, as in the case of BP, we associate each  $f_i$  with different factors from the joint probability distribution. This can be done by simply assigning the set of factors used at each step in VE to the corresponding factor in the clique tree. In the considered example, one possible assignment is thus the following:

$$\begin{aligned} f_1(A, B, C, D) &= p(A) p(B|A) p(C|A, B) p(D|A) \\ f_2(C, D, E) &= p(E|C) \\ f_3(D, E, F) &= p(F|D, E) \\ f_4(F, G) &= p(G|F) \end{aligned} \quad (15)$$

To calculate the unconditional probabilities in the JT we use the same message passing scheme as in BP. The only difference with respect to BP is that the scopes of the messages from nodes to factors and from factors to nodes, given in equations (11) and (12), depend on multiple random variables rather than just one. For example, the scope of the message from cluster node  $A, B$  to  $f_1$  is  $A, B$ , i.e.  $\mu_{AB, f_1}(A, B)$ . The list with all the messages that are needed to calculate the marginal probabilities are shown in the Supplementary Material.

The unconditional joint probability for the variables in cluster node  $\mathbf{X}_s$ , provided that the graph is a BN, is given by:

$$p(\mathbf{X}_s) = \prod_{f_j \in \mathbf{F}_s} \mu_{f_j, \mathbf{X}_s}(\mathbf{X}_s) \quad (16)$$



where  $\mathbf{F}_s$  are the factor nodes in the neighbourhood of the cluster node  $\mathbf{X}_s$ . To calculate the marginal probability for one random variable in the set  $\mathbf{X}_s$  from  $p(\mathbf{X}_s)$ , we just sum over the rest of the variables in  $\mathbf{X}_s$ . For example, the expression to calculate  $p(C)$ , the probability that an attacker obtains user privileges in Host 1, is given by:

$$p(C) = \sum_D p(C, D) = \sum_D \mu_{f_1, CD} \cdot \mu_{f_2, CD} \quad (17)$$

Evidence of compromise can be included in the inference using the JT algorithm in the same way as in BP, as previously described.

Finally, as in the case of VE, the computational complexity of the JT algorithm is exponential in the scope of the biggest factor in the clique tree. Concretely, if all the variables in the graph are discrete and have  $K$  possible values each (in the case of the BAG,  $K = 2$ ), JT scales in time and space as  $\mathcal{O}(|F|K^s)$ , where  $|F|$  is the number of factors and  $s$  is the size of the scope of the largest factor in the clique tree. JT therefore suffers from the same scalability problems as VE. However, for JT, we only need to run VE to create the clique tree once and then compute and store all the messages, which is significantly more efficient than in the case of VE, where we need to run the algorithm from the start each time we want to compute the marginal probability for a single node.

## 5 EXPERIMENTS

In this section we present the experimental results comparing the performance of JT to that of the VE algorithm in the version used in [9]. More broadly the purpose of the experiments is to analyse the performance of VE and JT algorithms for inference in BAGs in terms of both time and the memory requirements to compute the marginal probabilities. This allows us to examine their suitability for static and dynamic analysis of AGs, since the BAG models proposed in the literature [9], [12], [22] have not been evaluated experimentally. Through these experiments we also want to highlight the differences between VE and JT when used for inference in BNs and the impact of an elimination ordering heuristic. For VE we establish the elimination ordering at random, since in [9] no elimination ordering algorithm is proposed, whereas for JT, we use the min-weight heuristic [26] to find the elimination ordering needed to build the clique tree. For the implementation of both algorithms we use the Bayes Net toolbox for Matlab<sup>2</sup>.

In order to provide a comprehensive evaluation of the algorithms we have used synthetic AGs for the experiments, as many graphs of different sizes and structure are needed to provide meaningful results. Given that generating AGs for real systems is far from trivial, such a broad range of AGs is not available empirically. The typical structures of AGs are not clear from the examples encountered in the literature, and we expect them to vary significantly since they depend on both the network topology and the number of vulnerabilities. Considering these limitations and, in order to give a good insight of the performance of VE and JT we have considered two kind of structures for our synthetic

AGs: Random graphs, where we control the in-degree of the nodes (which is related to the number of vulnerabilities that a node in the network can have), and cluster graphs, where we explore the behaviour of the algorithms with respect to the size of the clusters. For real scenarios we expect to have a range of possible structures between these two kind of graphs depending on the actual network topology. We also include in our discussion an example of AG generated using a realistic use-case representative of the corporate network of an SME.

The values of the probabilities of exploitation of vulnerabilities needed to build the conditional probability tables are drawn at random, as well as the kind of logical conditions to build these tables: AND/OR. It is important to note that this does not have an impact in the time or memory required to calculate the unconditional probabilities with VE and JT, but only in the values of the probabilities of the nodes.

### 5.1 AG example

For the first example, we have built a realistic small use-case scenario, to show the results of our proposed techniques in a understandable way. Later, we will show more tests performed on larger AGs generated synthetically.

In Figure 7, we have depicted a typical network for an SME. In detail, we have two internal LANs (one for finance/accounting, one for technicians), a Wireless LAN for visitors coming into the premises (but that, if compromised, can be also used to reach the internal network), and finally a DMZ hosting the SME servers (in the example, a public Web Server, a Mail Server, and a Local Database used to store public and private data). For each node, we show in Figure 7 the set of reachable ports, and from which node they are reachable (this includes those ports open/filtered by the firewall as well as those open/closed by local firewalls, switches, routers, etc.). In addition, we have highlighted some vulnerabilities that might be present on the network nodes. For each vulnerability, we have reported the port on which it can be exploited (if remote), the CVE identifier, the type of vulnerability (DoS, elevation of privilege, etc.), and the likelihood of exploiting it. We have based this likelihood on the CVSS Exploitability Subscore (divided by 10). Although, as discussed in Section 3, this is not sufficient, as other aspects need to be taken into account such as the resources/skills of the attackers and the knowledge of the existence of an exploit, among others, this is in line with existing use in the literature. In our particular example, when the CVSS Exploitability Subscore is 1.0 (which means, an exploit already exists and it is ready/easy to use), we have decided to lower the value to 0.95, since a probability of successful exploitation of 1.0 would mean that the attacker has already reach the next security state without necessarily exploiting the vulnerability, which is not true.

In Figure 8 we show the corresponding BAG where the goal of the attacker is assumed to be to compromise the Database Server. It is important to note that we have clustered all the (similar) users in one single node in the BAG, and similarly for the system administrators. From an AG perspective, compromising one or several machines with the same behaviour (in terms of connectivity, services running, and vulnerabilities) can be considered the same in

<sup>2</sup> The implementation is available at <https://github.com/bayesnet/bnt>

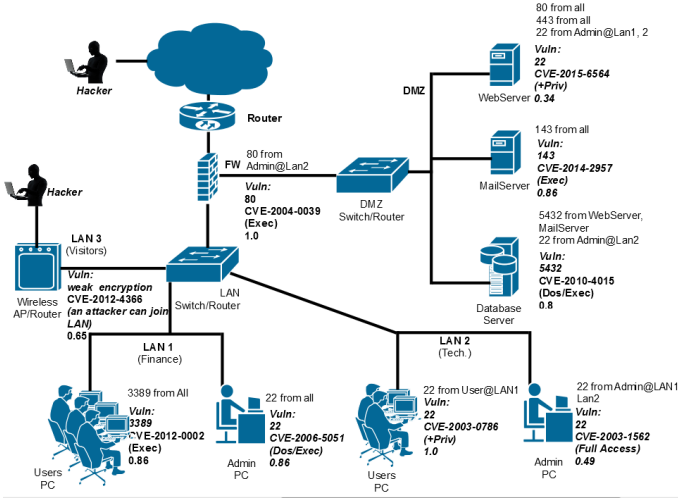


Fig. 7. Example of a network representative of an SME.

terms of privilege escalation to compromise a given target; although in terms of the potential damage that an attacker can cause to a network, (for example in terms of information leaks) the number of similar compromised machines can be very important. Furthermore, although clustering similar machines in our example is a simplification of the reality (we assume that the behaviour is exactly the same, which is not the real case in most corporate networks) the approximation still depicts in quite a realistic way the kind of AG we can produce for typical corporate networks.

In our example, we do not consider insider attackers, and we assume that an attacker can start an attack to the corporate network from the Internet or trying to access it illegally from the visitors Wireless LAN.

Using the JT algorithm, the time required to create the clique tree and to compute the marginal probabilities for each node in the BAG, shown in Figure 8, is less than 0.05 seconds on a common desktop computer, which makes JT suitable as a tool for the analysis of this AG.

Although this example intends to be representative of a real scenario in a small corporate network, in some practical situations we can expect more complex and much larger AGs. Then, to assess the suitability of JT for the analysis of AGs, in the rest of the experiments we use synthetic AGs with varying numbers of nodes and topologies, and compare the results with the VE algorithm proposed by [9].

## 5.2 AGs with pseudo-random structure

For this kind of graphs we build the BAG directly by generating random DAGs where we limit the maximum number of parents for each node. This corresponds to limiting the maximum number of vulnerabilities that can allow an attacker to obtain a certain security condition. In our opinion, this assumption is reasonable since, in real scenarios, we expect to have a reduced number of vulnerabilities that allow an attacker to compromise a node in the network. Then, for each node in the graph  $X_j$ , we randomly select the number of parents by drawing a random integer  $n_p$  in the interval  $[1, m]$  uniformly, where  $m$  is the maximum number of possible parents. After that, we randomly select

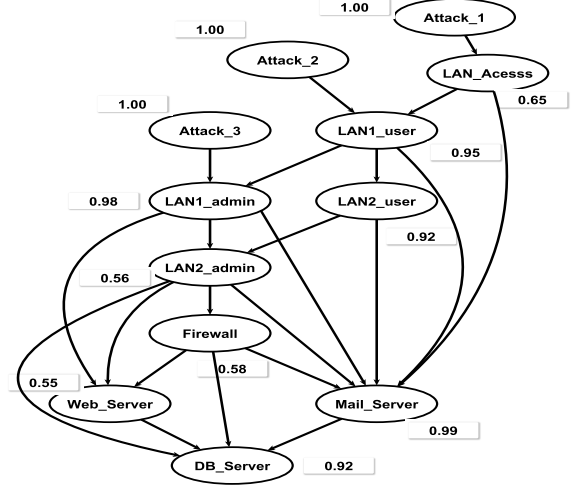


Fig. 8. BAG for the network in Figure 7 showing the unconditional probabilities that an attacker can successfully reach a security state for all the nodes in the BAG.

$n_p$  parent nodes for  $X_j$  from the set of nodes in the BAG for which  $X_j$  is not a parent node (to avoid directed cycles). For  $m$ , we have explored the values 2, 3, and 4. For the number of nodes  $n$  in the graph we have explored values in the range  $[10, 220]$ . However, depending on the algorithm and the value of  $m$  we have limited the value of  $n$  because of physical memory limitations. For example, VE could only be applied to random networks with  $m = 4$  up to  $n = 60$  nodes. For each value of  $n$  and  $m$ , we have generated 20 random networks and, for each network, we compute the unconditional probabilities for all the nodes with VE and JT.

In Figure 9 we show the average time required to compute all the unconditional probabilities for VE and JT. In the case of JT, the measured time includes the time required to find the elimination order, build the clique tree, compute all the messages, and calculate the marginal probabilities. It can be appreciated the remarkable difference in performance of both algorithms: JT is much faster than VE in all the cases considered. For example, for  $n = 130$  and  $m = 2$ , the average time to compute the unconditional probabilities is less than 1 second for JT, whereas for VE is almost 1000 seconds. As described in Section 4, JT only needs to build the clique tree and compute the message once, while in VE we have to compute everything each time we want to calculate the unconditional probability in one node of the network, which is much less efficient. Besides this, the elimination ordering also plays an important role in the time performance of the algorithms: a good elimination ordering algorithm can reduce the scope of the factors appearing in the clique tree (or the  $\phi_i$  factors in VE), and thus reduce the size of the tables that need to be computed. As we show next, this also has an impact in the memory needed for these inference algorithms.

The average number of nodes in the scope of the biggest factor for both, VE and JT, as a function of the number of nodes is shown in Figure 10. This corresponds to the number of variables of the biggest  $\phi_i$  and  $f_i$  for VE and JT respectively. We can appreciate, that, again, JT is more efficient than VE in all the cases. In this case, the difference between

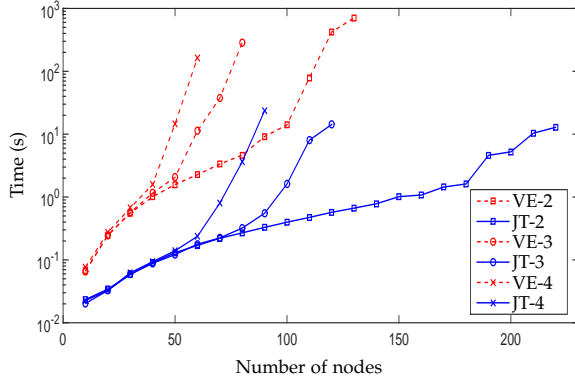


Fig. 9. Average time to compute the unconditional probabilities for VE and JT. The notation VE- $m$  and JT- $m$  stands for the value of  $m$ , the maximum number of parents of each node, used to generate the graphs in each case.

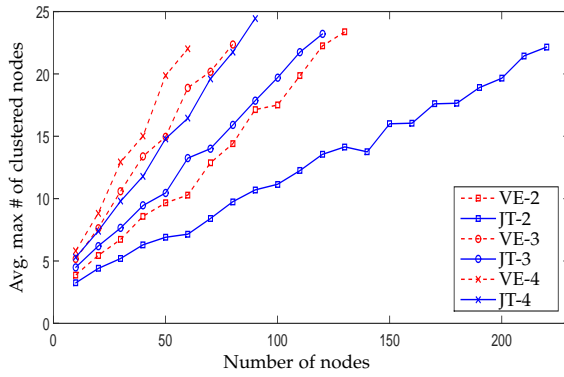


Fig. 10. Average number of nodes of the biggest factor for VE and JT. The notation VE- $m$  and JT- $m$  stands for the value of  $m$ , the maximum number of parents of each node, used to generate the graphs in each case.

results is only due to the elimination ordering algorithm. As mentioned before, we have not used any elimination ordering algorithm for VE, as none was proposed in [9], whereas for JT we are using the min-weight heuristic. In the case of using the same elimination ordering algorithm, we should expect similar results for VE and JT. Then, we can observe the computational savings in terms of memory (and then, also in time) when using an elimination ordering heuristic. It is important to note also that the tables needed to compute the factors in VE and JT grows exponentially with the number of variables in the scope of the factors. For example, for  $n = 130$  and  $m = 2$  the average number of nodes in the biggest factor for VE is approximately 23, whereas for JT is 14. This means that the average memory required to store these factors (considering that we use 8 bytes to store each entry in the table) is  $2^{23+3}/1024^2 = 64$  Megabytes for VE without an elimination ordering heuristic and only  $2^{14+3}/1024^2 = 0.125$  Megabytes for JT using the min-weight heuristic.

As described in Section 4, when evidence of compromise is observed in some of the graph nodes, the JT algorithm does not need to build the clique tree again but only to recompute the messages taking into account the evidence. In this sense, *static analysis* with the JT algorithm consists in building the clique tree, computing all the messages

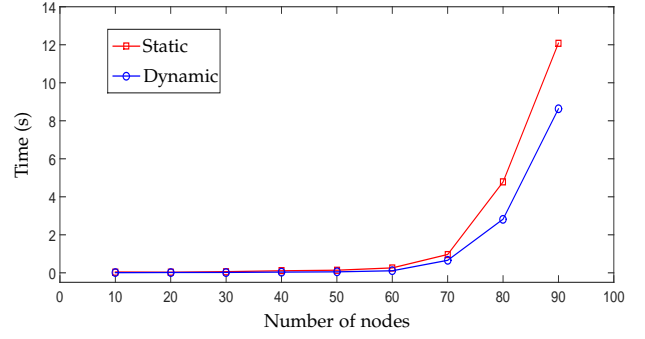


Fig. 11. Time to compute the unconditional probabilities with the JT for random networks with  $m = 4$  for the static the dynamic analysis (when we observe evidence at one node).

and calculating the unconditional probabilities in the absence of any evidence of compromise. In contrast, *dynamic analysis* consists only in recomputing all the messages and recalculating the marginal probabilities (conditioned on the evidence). This scenario is frequent in AG analysis by correlating with compromise evidence from IDS and Security Information and Event Management (SIEM) at run-time. Figure 11 shows the difference in time between the static and the dynamic analysis using the JT for graphs with  $m = 4$  and nodes  $n$  varying from 10 to 90. For the dynamic analysis we randomly select one node where we consider evidence has been observed. It is interesting to note that, although the time to recompute the probabilities is lower, the difference is not very significant. This means that, for these kind of pseudo-random graphs, the bottleneck of the JT algorithm is the computation of the messages rather than the elimination ordering algorithm applied to build the tree, since the number of variables for each factor is high (at least for some factors), as we can observe in Figure 10. This is due to the fact that the generated graphs are highly connected. This kind of analysis does not apply in the case of VE as the steps to compute the marginal probabilities are the same under the static and the dynamic analysis.

### 5.3 AGs with cluster structure

To generate synthetic graphs with a cluster structure, we have considered networks with clusters of the same size,  $n_c$ . Then, for each cluster, we have generated pseudo-random subgraphs using the same procedure as before, limiting the maximum number of parents for a given node to  $m$ . Finally, we have included dependencies between clusters by adding one edge from one node in each cluster to one node in other clusters provided that the added edge preserves the DAG structure required for BNs. For the first experiment with cluster based graphs we have set  $n_c = 10$  and varied the number of nodes in the network from 100 to 1000. Then, we have measured the time required to compute the unconditional probabilities using VE (with random elimination ordering) and JT (using min-weight to build the clique tree) for both the static and the dynamic analysis, considering that we observe evidence at one node. The results in Figure 12 show again a significant difference in performance between JT and VE, as in the case of pseudo-random graphs. This difference is mainly due to the fact

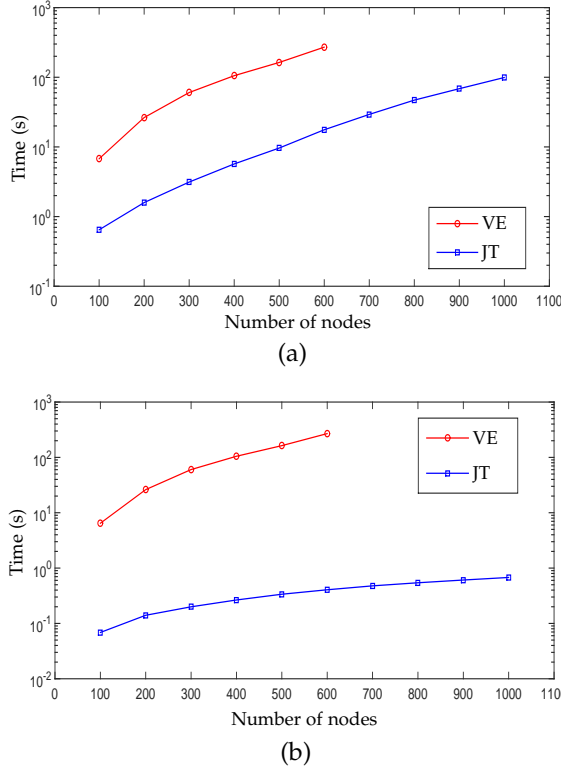


Fig. 12. Time to compute the unconditional probabilities for VE and JT for cluster networks with clusters of size 10 and  $m = 4$ : (a) for the static analysis and (b) for the dynamic analysis (when we observe evidence at one node).

that in VE we need to recompute everything each time we calculate the marginal probability for a node rather than the lack of an elimination ordering heuristic for VE.

We can observe in Figure 12 that the time to calculate the marginal probabilities required by VE for the static and the dynamic analysis is almost the same. This is not surprising since, as mentioned before, VE basically performs the same operations for both analyses. In contrast, we can appreciate a noticeable difference for JT between the measured time for the static and the dynamic analysis. In this case, the proportion of time required to build the clique tree is significantly higher than the proportion of time needed to compute all the messages and calculate the probabilities. Note that this is just the opposite behaviour from what we have observed for pseudo-random networks. This behaviour can be explained by the clustered structure of the graph, which renders smaller factors in the clique tree.

In this experiment we have omitted the results showing the memory used by the algorithms, as the cluster size, rather than the number of nodes, is the variable that mainly determines the size of the maximum number of variables for each factor in VE and JT. Thus the maximum number of variables for the biggest factor remains broadly constant with the number of nodes in the graph. However, as in the case of pseudo-random graphs, the use of min-weight heuristic to find an elimination ordering reduces significantly the memory required by JT compared to VE.

Finally, in Figure 13 we show the performance of JT for the static and the dynamic analysis in cluster graphs whilst varying the size of the clusters (from 10 to 50) and

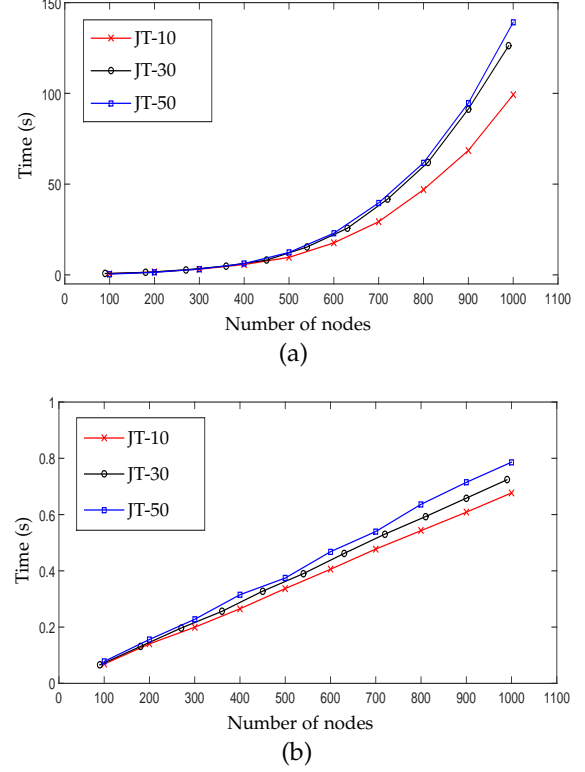


Fig. 13. Time to compute the unconditional probabilities for JT algorithm for cluster networks with different cluster sizes (10, 30, and 50) and  $m = 4$ : (a) for the static analysis and (b) for the dynamic analysis (when we observe evidence at one node).

the size of the graph (from 100 to 1000), setting  $m = 4$ . As in Figure 12, the difference in time between the two analyses is significant. Moreover we can appreciate a different behaviour: Whilst the time required for static analysis still scales exponentially with the number of nodes, that required for dynamic analysis broadly scales linearly with the number of nodes. The exponential behaviour of the static analysis is due to the elimination ordering algorithm used to build the clique tree. However, the clustered structure of the graph combined with the weak dependencies between clusters, result in factors of a similar size regardless of the size of the network. This allows the computation of the messages and of the probabilities to scale nearly linearly.

It is further interesting to highlight the reduced time (less than 1 second) required by JT to recompute the unconditional probabilities in the dynamic analysis when we observe evidence of compromise. This makes JT an appealing choice to perform dynamic analysis on real AGs, so that risk mitigation strategies can be applied. This contrasts with the very high cost of recomputing the same probabilities for VE, which renders it unusable for practical purposes. However, the exponential memory and time scalability of JT (and so, VE) for the static analysis, still limits the application of the algorithm to medium size graphs and is dependent of the topology of the AG.

The nearly linear scalability for the dynamic analysis of clustered BAGs with the JT algorithm reflects the importance of the granularity at which modelling with AGs is applied, and in particular considering the sub-network struc-

ture of corporate networks. For example, the AG model in [45] only considers vulnerabilities that let an attacker move from one subnetwork to another. From the AG perspective this means that compromises within a subnetwork are considered equivalent for the risk assessment of the whole network. In this sense, intermediate levels of granularity can be adopted to achieve better accuracy when modelling the AG. In any case, taking into account the cluster structure of networks in AG modelling is a key aspect to be considered to perform a tractable dynamic risk assessment of networks with BAGs using the JT algorithm.

## 6 EXTENSIONS OF THE BAYESIAN ATTACK GRAPH MODEL

The BAG model presented in Section 3 can be extended to take into account further aspects and represent more realistic scenarios.

The effect of zero-days vulnerabilities can be taken into account by adding one extra node (with no parents) connected to each of the security condition nodes in the graph. The difficulty lies in estimating a reasonable score to rate their probability of successful exploitation when building the conditional probability tables. Although in [12] they let the administrator of the network quantify the effect of the zero-days vulnerabilities, this is not a trivial task. The same reasoning applies when considering insider or social engineering attacks.

To model the uncertainty about the evidence that a node may be compromised, as derived from IDS and SIEM information, we should include correction terms in equations (3) and (4). One possible solution is to add one extra parent node for each existing node in the BAG, setting the prior probabilities of these added nodes to 1 (as in the case of the attacker's initial state node). Then, we can use the error probability of the IDS as a parameter to build the conditional probability tables of the corresponding nodes. The modification of (3) to compute the probability table in the AND case including this uncertainty results in:

$$p(X_i | \mathbf{pa}_i) = \begin{cases} p_e, & \exists X_j \in \mathbf{pa}_i | X_j = F \\ 1 - (1 - p_e) \times \\ (1 - \prod_{j: X_j} p_{v_j}), & \text{otherwise} \end{cases} \quad (18)$$

In the OR case, the modification of expression (4) is given by:

$$p(X_i | \mathbf{pa}_i) = \begin{cases} p_e, & \forall X_j \in \mathbf{pa}_i | X_j = F \\ 1 - (1 - p_e) \times \\ \prod_{j: X_j} (1 - p_{v_j}), & \text{otherwise} \end{cases} \quad (19)$$

where  $p_e$  is the error probability of the IDS.

In [10] a dynamic BN is proposed to consider variations of the temporal metrics in the CVSS scores. Although the theoretical model in [10] is correct, as the temporal metrics do not change rapidly over time, the extra level of complexity induced by the dynamic BN, which requires adding extra nodes to the network, limits the application of the model in real scenarios. It would perhaps be more reasonable, in terms of computational complexity, to update

the conditional probability distributions in the model we propose when there are changes in the CVSS scores. To support this claim, the experimental results in Section 5 show the computational cost implications of augmenting the complexity of the BAG.

## 7 CONCLUSIONS

We have proposed in this paper a BN model for AGs and efficient exact inference techniques for their static and dynamic analysis. These techniques allow to assess the security risk of the nodes in a network against cyber-attacks by calculating the probabilities that an attacker can compromise each node given the nodes that have been already compromised (if any). These calculations can help system administrators to choose the most appropriate countermeasures during an ongoing attack.

We have reviewed and proposed solutions to the shortcomings of existing BAG models in the literature, such as the implications of adding prior probabilities in the attacker's capabilities. In this sense, although some of the assumptions in our model are still restrictive, we have proposed and sketched some direct extensions of the Bayesian model that we will consider in our future work.

We have also shown the limitations of previous state-of-the-art techniques to perform static and dynamic analysis of BAG, and the importance of using efficient algorithms to calculate the marginal probabilities on the nodes. To support this, we have presented an extensive experimental evaluation with synthetic AGs to measure the time and memory required to calculate these marginal probabilities. This stands in contrast with related work in the literature where such analysis is missing. Our experimental results show the usefulness of the JT algorithm to perform static and dynamic analysis of AGs for graphs with hundreds of nodes, which in most cases would correspond to corporate networks of thousand of nodes (depending on the number of existing vulnerabilities). We have shown the important improvements of the JT algorithm in terms of time and memory with respect to the VE algorithm proposed in [9] to calculate the marginal probabilities in BAGs. The results indicate that VE is not suitable for the dynamic analysis of AGs, since the time required to recompute the marginal probabilities in the graph is too high, limiting the reaction time of system administrator to respond to an ongoing attack.

Finally, we have also shown the importance of modelling AGs with a higher level of granularity, taking into account the subnetworks structure of typical corporate networks. The network clustering enables the dynamic analysis to become tractable and scales linearly in the number of nodes. This is important, particularly as evidence of compromise from IDS and SIEM information may arrive at a fast pace.

Further research directions include to explore more scalable inference techniques that allow to perform dynamic analysis of AGs in larger networks, to extend the BAG model to make it less restrictive, and to investigate more accurate mechanisms to estimate the probability of exploitation of vulnerabilities rather than using CVSS scores. Furthermore, beyond the static and dynamic analysis, we can extend our BAG model to other purposes, such as to



prioritise forensic investigations, given the usefulness of AGs for this task [46].

## ACKNOWLEDGMENTS

This work has been supported by the UK government under EPSRC grant EP/L022729/1. The authors would like to thank British Telecom for their collaboration in this research and our colleagues in our research group for their contribution to this work through many useful discussions.

## REFERENCES

- [1] Gartner, Inc., "Gartner Says Worldwide Information Security Spending Will Grow Almost 8 Percent in 2014 as Organizations Become More Threat-Aware," <http://www.gartner.com/newsroom/id/2828722>, 2014.
- [2] Information is Beautiful, "World's Biggest Data Breaches," <http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/>, 2015.
- [3] E. Wheeler, *Security Risk Management: Building an Information Security Risk Management Program from the Ground Up*, Syngress Publishing, 2011.
- [4] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Procs. of the IEEE Symp. on Security and Privacy*, 2002, pp. 273–284.
- [5] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Procs. 15th IEEE Workshop on Computer Security Foundations*, 2002, pp. 49–63.
- [6] M. Albanese, S. Jajodia, and S. Noel, "Time-efficient and cost-effective network hardening using attack graphs," in *Int. Conf. on Dependable Systems and Networks*, 2012, pp. 1–12.
- [7] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modelling modern network attacks and countermeasures using attack graphs," in *Conf. Computer Security Applications*, 2009, pp. 117–126.
- [8] A. Barth, B.I.P. Rubinstein, M. Sundararajan, J.C. Mitchell, D. Song, and P.L. Bartlett, "A learning-based approach to reactive security," *IEEE Trans. on Dependable and Secure Computing*, vol. 9, no. 4, pp. 482–493, 2012.
- [9] Y. Liu and H. Man, "Network vulnerability assessment using Bayesian networks," in *Data Mining, Intrusion Detection, Inform. Assurance, and Data Networks Security*, 2005, vol. 5812, pp. 61–71.
- [10] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring network security using dynamic Bayesian network," in *Procs. 4th Workshop on Quality of Protection*, 2008, pp. 23–30.
- [11] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *Procs. 22nd IFIP WG 11.3 Conf. on Data and Applications Security*, 2008, pp. 283–296.
- [12] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Trans. on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.
- [13] K.P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT press, Cambridge, MA, 2012.
- [14] L. Rabiner and B.H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.
- [15] C. Phillips and L.P. Swiler, "A graph-based system for network-vulnerability analysis," in *Procs. of the Workshop on New Security Paradigms*, 1998, pp. 71–79.
- [16] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *Formal Methods for Components and Objects*. Springer, 2004, pp. 344–371.
- [17] L.P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," in *Procs of the DARPA Inform. Survivability Conf. and Exposition II*, 2001, vol. 2, pp. 307–321.
- [18] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Procs. 9th Conf. on Computer and Communications Security*. ACM, 2002, pp. 217–224.
- [19] S. Jajodia, S. Noel, and B. OBerry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats*, pp. 247–266. Springer, 2005.
- [20] X. Ou, W. F. Boyer, and M.A. McQueen, "A scalable approach to attack graph generation," in *Procs. 13th Conf. on Computer and Communications Security*, 2006, pp. 336–345.
- [21] M. Frigault and L. Wang, "Measuring network security using Bayesian network-based attack graphs," in *Procs. 3rd IEEE Int. Workshop on Security, Trust, and Privacy for Software Applications*, 2008, pp. 698–703.
- [22] M. Frigault, "Measuring Network Security using Bayesian Network-based Attack Graphs," M.S. thesis, Concordia University, Montreal, Canada, 2010.
- [23] R. Dechter, "Bucket elimination: A unifying framework for probabilistic inference," in *Procs. 12th Int. Conf. on Uncertainty in AI*. Morgan Kaufmann, 1996, pp. 211–219.
- [24] Common Vulnerability Scoring System, V3, "Development update," <https://www.first.org/cvss>, 2015.
- [25] S.H. Houbb and V.N.L. Franqueira, "Estimating ToE Risk Level Using CVSS," in *7th Int. Conf. on Availability, Reliability, and Security*. IEEE, 2009, pp. 718–725.
- [26] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT press, Cambridge, MA, 2009.
- [27] G.F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *J. of AI*, vol. 42, no. 2, pp. 393–405, 1990.
- [28] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. on Algebraic Discrete Methods*, vol. 8, no. 2, pp. 277–284, 1987.
- [29] N.L. Zhang and D. Poole, "A simple approach to Bayesian network computations," in *Procs. 10th Canadian Conf. on AI*, 1994, pp. 171–178.
- [30] M. Fishelson and D. Geiger, "Optimizing exact genetic linkage computations," *J. of Computational Biology*, vol. 11, no. 2-3, pp. 263–275, 2004.
- [31] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *Int. J. of Approximate Reasoning*, vol. 15, no. 3, pp. 225–263, 1996.
- [32] U. Kjærulff, "Triangulation of Graphs—Algorithms Giving Small Total State Space," Tech. Rep. R90-09, Department of Mathematics and Computer Science, Aalborg University, Denmark, 1990.
- [33] B. Schneier, "Attack trees," *Dr. Dobbs journal*, vol. 24, no. 12, pp. 21–29, 1999.
- [34] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Procs. of the National Conf. on AI*, 1982, pp. 133–136.
- [35] J.H. Kim and J. Pearl, "A computational model for causal and diagnostic reasoning in inference systems," in *Procs. 8th Int. Conf. on AI*, 1983, pp. 190–193.
- [36] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.
- [37] C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, NY, 2006.
- [38] B.J. Frey, *Graphical Models for Machine Learning and Digital Communication*, MIT press, 1998.
- [39] F.R. Kschischang, B.J. Frey, and H.A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Inform. Theory*, vol. 47, no. 2, pp. 498–519, 2001.
- [40] P.P. Shenoy and G.R. Shafer, "Axioms for probability and belief-function proagation," in *Procs. 4th Conf. on Uncertainty in AI*, 1990, pp. 169–198.
- [41] G.R. Shafer and P.P. Shenoy, "Probability propagation," *Annals of Mathematics and AI*, vol. 2, pp. 327–352, 1990.
- [42] S.L. Lauritzen and D.T. Spiegelhalter, "Local computations with probabilities on graphical structures and their applications to expert systems," *J. of the Royal Statistical Society. Series B (Methodological)*, vol. 50, no. 2, pp. 157–224, 1988.
- [43] F.V. Jensen, K.G. Olesen, and S.K. Andersen, "An algebra of Bayesian belief universes for knowledge-based systems," *Networks*, vol. 20, no. 5, pp. 637–659, 1990.
- [44] V. Lepar and P.P. Shenoy, "A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for computing marginals of probability distributions," in *Procs. 14th Conf. on Uncertainty in AI*, 1998, vol. 14, pp. 328–337.
- [45] S. Jajodia, S. Noel, P. Kalapa, M. Albanese, and J. Williams, "Cauldron mission-centric cyber situational awareness with defense in depth," in *Military Communications Conf.*, 2011, pp. 1339–1344.
- [46] C. Liu, A. Singhal, and D. Wijesekera, "Creating integrated evidence graphs for network forensics," in *Advances in Digital Forensics IX*, 2013, pp. 227–241.